Design and Implementation of Web Forward Proxy with Shibboleth Authentication

AUTHOR(S):

KOMURA, Takaaki; SANO, Hiroaki; DEMIZU, Noritoshi; MAKIMURA, Ken

CITATION:

KOMURA, Takaaki ...[et al]. Design and Implementation of Web Forward Proxy with Shibboleth Authentication. 11th Annual International Symposium on Applications and the Internet, SAINT 2011, Munich, Germany, 18-21 July, 2011, Proceedings 2011: 321-326

ISSUE DATE:

2011-07-21

URL:

http://hdl.handle.net/2433/147168

Design and Implementation of Web Forward Proxy with Shibboleth Authentication

KOMURA Takaaki
Institute for Information Management and Communication
Kyoto University, Kyoto, Japan
Email: komrua@media.kyoto-u.ac.jp

SANO Hiroaki
Kyoto University Library, Japan
Email: zuoye@kulib.kyoto-u.ac.jp

DEMIZU Noritoshi
OCTOPATH Corporation, Kyoto, Japan
Email: demizu@octopath.co.jp

MAKIMURA Ken
OCTOPATH Corporation, Kyoto, Japan
Email: macky@octopath.co.jp

***Abstract*—We propose a web forward proxy server with authentication method using Shibboleth. With this proxy Single Sign-On would benefit a user and also authentication using Shibboleth protocol solves problems in basic access authentication and digest access authentication supported by existing web forward proxy servers. In order to realize it, the proxy needs to recognize attributes of shibboleth protocol and session cookies and to modify session cookies. We implemented system and evaluated it by accessing to electronic journals from a test network.**

***Keywords*-Security Assertion Markup Language (SAML); Shibboleth; Single Sign-On (SSO); Web Forward Proxy**

I. Introduction

Web proxy servers are commonly used to access from web browser on an intranet to the web servers on the Internet. A proxy server has a large variety of potential purposes, however, in this paper we focus on the following two points in case the proxy requires user authentication and logs user's accesses.

- Rapid incident response
  e.g. Operators in the intranet could find easier which client tries to attack to a web server on the Internet.
- Web access statistics
  e.g. The log shows us that the number of web access from individual department in the organization.

These features are often important for large organization.

Existing forward proxies support only basic access authentication [1], [2] (Basic Auth) and digest access authentication [2], [3] (Digest Auth) as a user authentication method. We propose a web forward proxy which supports an authentication by using Shibboleth [4] protocol. The proxy provides Single Sign-On (SSO) access for users. Furthermore the Shibboleth authentication solves security problems in Basic Auth and Digest Auth.

We describe types of proxies and problems in Basic and Digest Auth and our motivation in Section II. A Summary of Shibboleth is described in Section III. We explain a design and an implementation of our proposal web proxy with Shibboleth authentication method in Section IV. Proposed proxy is just going to be evaluated on the network in our university in order that students and staffs in our university access electronic journals (E-Journals) on the Internet. We describe experiments to access to E-Journal and an evaluation in Section V.

II. Proxy and Authentication

A. Types of Web Proxy

The proxies are classified into two types of behavior: forward proxies and reverse proxies. In this section, we describe that forward proxies are appropriate for our purposes after showing the differences of them.

Forward proxy server is need to be set up for web browser by one of following three ways:

- directly specify pairs of hostname and port number of forward proxy
- specify URIs of Proxy Auto-Configuration (PAC) file written in JavaScript
- use Web Proxy Auto-Discovery (WPAD) [5] protocol to find PAC file automatically

Browser connects to the forward proxy and requests to access contents on target web server using URI. Also the browser chooses which forward proxy to be used or not to be used according to target URIs which imply hostname, port number, protocol, etc when it uses PAC. Forward proxy relays the request without change from the web browser to the web server specified in URI.

Reverse proxy server appears to web browser to be an ordinarily web server. Web browser (or user) connects to the reverse proxy explicitly targeted by URI. The reverse proxy parses the URI and forwards to one or more origin servers which handle the requested URI. The response is returned as if it came directly from the reverse proxy.

A forward proxy is usually situated between the client and the server(s) hosting the desired resources, and a reverse proxy is usually situated closer to the origin server(s) and generally only return a configured set of resources. Reverse proxy is typically used for hiding servers, load balancing,

site integration, SSL acceleration, etc. The reverse proxy and their origin servers are often belong the same organization.

The biggest advantage of forward proxy over reverse proxy is that URL is not rewritten. Therefore, users are not annoyed by URL rewritings and administrators need not keep watching changes of contents and hostnames of origin servers to keep URL rewritings correct. The disadvantages are that users need set up proxy in browser and that standard authentication methods are limited to basic and digest authentications.

B. Authentication

Both forward proxies and reverse proxies support Basic Auth and Digest Auth for user authentication. Their authentication schemes have security problems respectively as shown later in II-B1 and II-B2.

Reverse proxies can utilize additional authentication schemes which are called Form Authentication, Client Certificate Authentication on HTTPS and so on. Although reverse proxies have advantages in authentication, they have disadvantages that all URIs need to be converted suitable for reverse proxy.

We consider safe authentication for the forward proxy in this paper.

1) Problem in Basic Auth: Basic access authentication is a method designed to allow a web browser to provide credentials which forms from a user name and a password when making a request.

Before transmission, the user name and the password are appended with a colon. The resulting string is encoded with the Base64 algorithm.

$$credential = Base64(username : password)$$

For example, browser sends the user name '`Aladdin`' and password '`open sesame`', it would use '`QWxhZGRpbjpvcGVuIHNlc2FtZQ==`' for the credential. Where the string is Base64 encoded of '`Aladdin:open sesame`'.

The Base64 algorithm is not encryption and security is not the intent of the encoding step. This scheme is not considered to be a secure method of user authentication, the user name and the password are passed over the network as same as cleartext.

Even if a browser uses SSL (HTTPS) to connect to a web server, the credential for a proxy is not encrypted when the proxy requires user authentication, because sending the credential to the proxy has been done before starting encryption by SSL between the web browser and the web server.

2) Problem in Digest Auth: Digest access authentication is a method a web server can use to negotiate credentials with a web browser. It uses encryption by MD5 cryptographic hashing to send a password over the network and is safer than the Basic Auth. A *response* of Digest Auth in HTTP/1.0 is formed as follows [1]: [1]

$$\begin{aligned} response &= MD5(MD5(A1) : nonce : MD5(A2)) \\ A1 &= username : realm : password \\ A2 &= method : digestURI \end{aligned}$$

Note that a raw password is needed to generate *response* on web server. In other words, the web server has to access a raw password itself or a value of *MD5(A1)*. Many directory systems stored a hash of password instead of user's raw password and these systems can not use Digest Auth.

Though other systems which stored a raw password can support it, security risks come in when web servers are able to access user's raw passwords. To decrease the security risk, a directory system may generates a value of *MD5(A1)* and the web server accesses it instead of a raw password, however, operating expense increases.

C. Necessity of Web Proxy in our Environment

There are three major reasons of necessity of web proxy at Kyoto University.

The first one is gateway from private network to servers on the Internet. Most part of the network in Kyoto University uses private IP addresses and NAT routers are not provided for HTTP and HTTPS because of administrative reasons. The forward proxies are provided for these protocols.

The second is for rapid incident response. When network administrators receive incident report from site on the Internet, they must analyze log files on the proxy server to find a person whose accesses are regarded as attacks, guide the person not to do again and respond to the report rapidly. For example all of our user can not access the E-Journal site till the E-Journal site accepts our response, because the E-Journal site rejects accesses from whole network of our university. User's continuous downloading (intentional and unintentional) from E-Journal cause almost all incidents in our experience. There are a few incidents per year.

The last one is for statistics of web access. In our university, license fee of E-Journals are planning to be charged for departments depending on the number of downloading papers and documents from E-Journal sites by users in each department. We need statistics information about downloading by our users, however, few E-Journal sites give us them. Thus the proxy is needed to authenticate a user and to log about who download papers.

There are two types of forward proxy are installed in our university. One is anonymous forward proxy for accessing almost all web sites on the Internet. The other is Squid [6] proxy which required digest authentication for accessing E-Journal sites. Hence a PAC file is provided for uses to automatically select which type of proxy to use.

[1]The algorithm to generate *response* is more complex in HTTP/1.1. [2], [3] and details of it are omitted here.

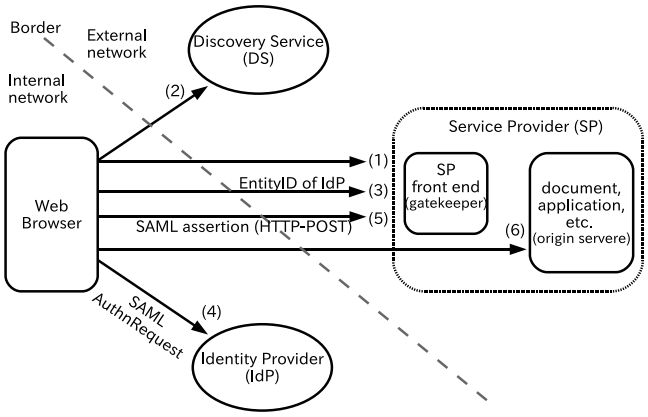

Figure 1. Flow of Shibboleth federation

So, our remaining concern is authentication methods. We need to record access logs with user ID for accounting and trouble-shooting. But we do not want to keep thousands of raw passwords in forward-proxy for administrative and security reasons. This is our motivation to develop web forward-proxy with Shibboleth authentication.

III. Shibboleth Federation

The Shibboleth enables secure access control for web-based applications and provides web single sign-on functionality. The Shibboleth includes three major software components: the Shibboleth Identity Provider (IdP), the Shibboleth Service Provider (SP) and the Discovery Service (DS). These three components are deployed separately but work together to provide secure access to Web-based resources. The IdP uses a web sign-on method which is a form authentication, a client certificate authentication, etc. The Shibboleth SP is a authentication module which behaves a wrapper for original web server which is called back-end server. After authentication at IdP, the IdP sends the message some security information called an "assertion" that proves the user signed on to the SP. Note that the assertion includes no password and it lowers the security risk. The DS asks a user to specify a home institution, then directs the user to that institution's IdP to authenticate.

We summarize here the process of sign-on using a shibboleth IdP and a shibboleth SP. Here is the typical procedure of the Shibboleth SSO.

1) Browser sends an HTTP request to access content/application in SP. SP checks whether valid Shibboleth session cookie [7] exists. If Shibboleth session has been established, goto the latter part of step 6. Otherwise, SP redirects the request to DS.
2) DS shows a form to select an IdP in federation. When user chooses his/her IdP, DS redirects the request back to SP.
3) SP redirects the request to selected IdP with SAML AuthnRequest issued by SP.
4) IdP authenticates user by username/password, physical authentication key, etc. Then, it sends SAML assertion back to SP by redirection or HTTP-POST.
5) SP checks the received SAML assertion. If it is valid and the user is allowed to access the content/application, the request is redirected again back to the original URL with the Set-Cookie header to set Shibboleth session cookie.
6) Browser sends an HTTP request to access the same URL with 1 accompanied by Shibboleth session cookie. While the session cookie is valid, content or application is provided for user.

IV. Design and Implementation

In this section, we propose a revised procedure of Shibboleth SSO for Shibboleth-capable forward-proxy.

Shibboleth SSO procedure was originally designed for content/application server. In order to apply it to forward-proxy server, the role of Service Provider (SP) in the original design is split into two; origin server (OrS) and gatekeeper (GK). Origin server is a term in the HTTP/1.1 specification [3] and provides content and/or application. GK, which consists of Policy Decision Point (PDP) and Policy Enforcement Point (PEP), resides at origin server in the original design, while GK resides in forward-proxy in our proposal. GK checks Shibboleth session cookie and decides whether the HTTP request should be accepted, rejected, or redirected to DS or IdP.

Our goal in designing a revised procedure for Shibboleth-capable forward-proxy are as following:

- It works with existing DS, IdP and browser implementations. Browser does not show uneasy message in the procedure.
- The changes to existing Shibboleth SP implementation is small so that it is easy to catch up future releases of Shibboleth SP and easy to review codes to keep secure and stable. Modified version of Shibboleth SP supports both the original procedure and our proposed procedure. Any existing feature of Shibboleth SP must not be broken.
- It is easy to configure DS, IdP, forward-proxy and PAC. Configurations of DS and IdP do not include information on proxyed origin servers.

In order to achieve these goals, GK takes the responsibility on behalf of origin servers to conduct browser to establish Shibboleth session by interacting with DS and IdP.

When GK as a forward-proxy finds an HTTP request without valid Shibboleth session cookie, it redirects the HTTP request to a special URL whose host part is the hostname of GK. Hence, the redirected HTTP request is processed by GK as a web server, which acts as if it is SP in the original procedure. That is, if redirected HTTP request to it does not contain valid Shibboleth session cookie associated with it, it initiates Shibboleth authentication by

redirecting the request to DS, then to IdP. When Shibboleth session cookie associated with it is set to browser after authentication, it diverts browser to a phantom URL whose host part is the same as the one in the original request URL and path part is a specially crafted phantom path. Phantom URL is processed by GK as a forward-proxy, which sets Shibboleth session cookie associated with origin server to browser. Then, the HTTP request is redirected back to the original request URL. While the Shibboleth session cookie is valid, GK as a forward-proxy forwards HTTP requests to the origin server.

Because our proposal requires forward-proxy to observe all HTTP requests, our current proposal cannot be applied to HTTPS sessions.

The detail of process of our proposal is described bellow and in figure 2.

1) a) Browser sends an HTTP request to access content/application in origin server (OrS) through forward-proxy. Gatekeeper (GK) in forward-proxy checks whether valid Shibboleth session cookie exists. If Shibboleth session has been established, goto the latter part of step 6. Otherwise, GK as a forward-proxy redirects the request to a special URL whose host part is the hostname of GK.
   b) GK as a web server receives the redirected request. If the request contains valid Shibboleth session cookie associated with GK as a web server, goto the latter part of step 5a. Otherwise, the request is redirected to DS. In the query string of the redirected HTTP response, "entityID" contains the hostname of GK and "return" contains the URL of SessionInitiator in GK as a web server.
2) DS shows a form for user to select an IdP in federation. After user chooses his/her IdP, DS redirects the request back to GK as a web server, since "return" is specified so in step 1a.
3) GK as a web server receives the response from DS. GK checks whether valid Shibboleth session cookie associated with GK as a web server exists. If it does, goto the latter part of step 5a. Otherwise, GK redirects the request to the selected IdP with SAML AuthnRequest issued by GK. The "destination" attribute in this SAML AuthnRequest specifies the URL of Assertion Consumer Service (ACS) in GK.
4) IdP authenticates user by username/password, physical authentication key, etc. Then, it sends SAML assertion to GK by using HTTP redirection or HTTP-POST as specified in SAML AuthnRequest.
5) a) GK checks received SAML assertion. If it is valid and the user is allowed to access the content/application, Shibboleth session is established and session key is generated.

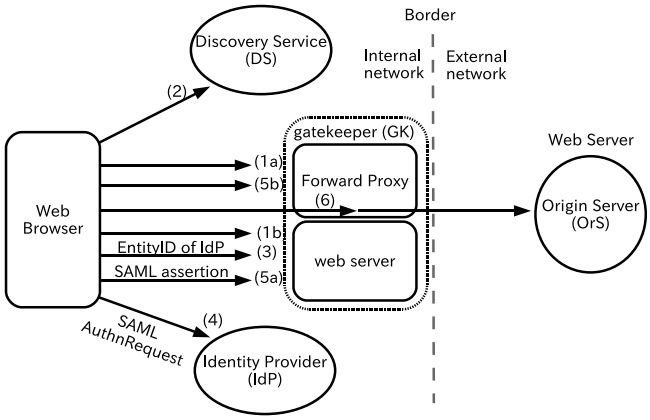

Figure 2. Flow of federation proposed proxy

Then, the request is redirected to a phantom URL under the hostname name of the original request URL with specially crafted path part followed by a temporary key in its query string. The redirected HTTP response contains the Set-Cookie header to set Shibboleth session cookie which will be checked at step 1band 3.
   b) GK as a forward-proxy intercepts the HTTP request to phantom URL. If it succeeds to restore session state corresponding to the temporary key in its query string, GK as a forward-proxy redirects the request to the original request URL. The redirected HTTP response contains the Set-Cookie header to set the same Shibboleth session cookie associated with GK as a web server.
6) Browser sends an HTTP request to access the same URL with 1a accompanied by Shibboleth session cookie.
   While Shibboleth session cookie is valid, GK as a forward-proxy forwards HTTP requests to OrS. GK as a forward-proxy removes Shibboleth session cookie before forwarding HTTP requests to avoid possible confusion in OrS.

Note that the browser has to be able to access the IdP and DS without authentication because an infinite loop occurs if authentication is required when the browser accesses to the IdP or DS. We solved the problem by using a PAC. The PAC written in JavaScript indicates which proxy or no proxy should be used according to the target URIs. We indicate by PAC that a browser uses the proposed proxy only when it accesses to origin servers that we expected to access with authentication.

Table I shows an example of authentication processes of ordinary Shibboleth and proposed forward proxy with Shibboleth. Note that "302" and "200", which are HTTP status codes, mean "redirection" and "OK (successful)"

Table I
THE COMPARISON OF AUTHENTICATIONS PROCESSES OF ORDINARY SHIBBOLETH WITH PROPOSED FORWARD PROXY WITH SHIBBOLETH

State		Ordinary Shibboleth	Proposed Forward-Proxy
1) a)	Browser → OrS	GET http://ors.net/doc	GET http://ors.net/doc *(forward-proxy intercepts the request and responses ..)*
	Browser ← OrS	*(GK in OrS intercepts the request and responses ..)*	302 https://proxy.net/Shibboleth.sso/Proxy/proxy.net?state=xxxx Set-Cookie: _shibstate_<32bit>=<OriginalURI>
1) b)	Browser → GK		GET *(above URL)*
	Browser ← GK	302 https://ds.net/DS/WAYF? entityID=http://ors.net/shibboleth-sp& return='http://ors.net/Shibboleth.sso/DS&..' Set-Cookie: _shibstate_<32bit>=<OriginalURI> *(GK is in OrS. "entityID" indicates OrS)*	302 https://ds.net/DS/WAYF? entityID=https://proxy.net/shibboleth-sp& return='https://proxy.net/Shibboleth.sso/DS&..' *("entityID" indicates GK as a the proxy)*
2)	Browser → DS	GET *(above URL)*	GET *(above URL)*
	Browser ← DS	302 http://ors.net/Shibboleth.sso/DS? entityID=https://idp.net/idp/shibboleth	302 https://proxy.net/Shibboleth.sso/DS? entityID=https://idp.net/idp/shibboleth
3)	Browser → GK	GET *(above URL)* *(GK is in OrS. OrS receives the request)*	GET *(above URL)* *(GK as a web server receives the request)*
	Browser ← GK	302 https://idp.net/idp/profile/SAML2/Redirect/SSO? SAMLRequest='<samlp:AuthnRequest..>' RelayState="cookie:<32bit>"	302 https://idp.net/idp/profile/SAML2/Redirect/SSO? SAMLRequest='<samlp:AuthnRequest..>' RelayState="cookie:<32bit>"
4)	Browser → IdP	GET *(above URL)*	GET *(above URL)*
	Browser ← IdP	200 *(HTML and SAML assertion)*	200 *(HTML and SAML assertion)*
	Browser ← IdP	*(Browser POST the assertion immediately by JavaScript)*	*(Browser POST the assertion immediately by JavaScript)*
5) a)	Browser → GK	POST: http://ors.net/Shibboleth.sso/SAML2/POST Cookie: _shibstate_<32bit>=<OriginalURI> SAMLResponse=<?xml..><samlp:Response.....>	POST: https://proxy.net/Shibboleth.sso/SAML2/POST SAMLResponse=<?xml..><samlp:Response.....>
	Browser ← GK	302 http://ors.net/doc (Original URI) Set-Cookie: _shibsession_<OrS-ID>=<128bit> Set-Cookie: _shibstate_<32bit>=""	302 http://ors.net/Shibboleth.sso/Proxy/proxy.net?state=xxxx Set-Cookie: _shibsession_<GK-ID>=<128bit>
5) b)	Browser → OrS		GET *(above URL)* *(the forward-proxy intercepts the request and responses ..)*
	Browser ← OrS		302 http://ors.net/doc (Original URI) Set-Cookie: _shibsession_<GK-ID>=<128bit> Set-Cookie: _shibstate_<32bit>=""
6)	Browser → OrS	GET http://ors.net/doc Cookie: _shibsession_<OrS-ID>=<128bit>	GET http://ors.net/doc Cookie: _shibsession_<GK-ID>=<128bit>
	Browser ← OrS	200 *(requested HTML)*	200 *(requested HTML)*

respectively. The careful reader may have noticed that our proposed procedure is almost the same with the original procedure except step 1b and 5b. In both procedures, GK takes the responsibility to establish and to maintain Shibboleth session. The only difference there is that GK resides whether in forward-proxy or at origin server. This similarity will result the change to the current Shibboleth SP implementation small.

Our current implementation is based on Shibboleth SP 2.4.2 and Apache http 2.2.17 running on Red Hat Enterprise Linux Server release 5.6. All changes are made to Shibboleth SP (written in C++) and our patch in the "diff -u" format is approximately 880 lines.

V. EXPERIMENTS AND EVALUATION

In our experiments, we visited several EJ/DB sites by using five popular browsers (IE8, Safari, Firefox, Opera and Chrome) through shibboleth-capable forward-proxy (abbreviated to "shibproxy" in this section) located in the network of Kyoto University. We prepared PAC which directed browser to shibproxy for restricted-access EJ/DB sites and to university's official forward-proxy for other sites.

When a user accesses a restricted-access EJ/DB site, his/her request is redirected to Discovery Service, where the user selects his/her IdP from two IdPs; the official IdP of Kyoto University or the special IdP for guest users of Kyoto University Library. Once the user is authenticated by IdP, the user is allowed to access any restricted-access EJ/DB sites without additional authentication.

Through our experiments, we found cases where browsers cannot retrieve some of resources in some pages. The cause of the problems is that there are cases where browsers do not accept cookies issued by shibproxy. Here are some examples and solutions of those problems.

Ex.1 Some EJ/DB sites use multiple host names such as www.example.org and portal.example.org. In such case, our PAC directs browser to shibproxy for any host names under example.org. Such sites often use resources of multiple sibling servers under example.org domain, such as assets.example.org and analytics.example.org, to construct one page. It is necessary for browser to accept cookies of those sibling servers to retrieve all resources referred by such pages through shibproxy. When browser is configured to block third-party cookies, four of five browsers accept cookies of sibling servers and successfully show such pages while one browser does not accept them and imperfectly shows such pages due to the lack of some of images, JavaScript, CSS, etc. To solve this problem, shibproxy can be configured to send Set-Cookie header with domain=.example.org attribute.

Ex.2 Some EJ/DB providers use multiple domain names to provide various kinds of EJ/DB. Their pages sometimes refer to resources of third-party EJ/DB sites with completely different host names but managed by the same owner. When browser is configured to block third-party cookies, all browsers do not accept cookies of those third-party sites and they cannot access third-party resources. Since the number of such third-party resources is limited, it can be solved by configuring shibproxy to accept HTTP requests for those third-party URLs.

Ex.3 Some browsers seem not to send cookies to retrieve some resources described in the HTML <link> element especially when those resources are loaded at the same time with the main HTML page. For example, some browsers seem not to send cookies to retrieve "favicon.ico". Hence, shibproxy is configured to forward HTTP request whose URL matches to "/favicon\w*.ico$" to origin server regardless whether Shibboleth session cookie exists or not. For another example, some browsers seem not to send cookies to retrieve OpenSearch [8] description. Since their file names vary, we have not solved this problem.

VI. Conclusion and Future Works

We proposed a web forward proxy server worked as the Shibboleth SP. The proxy supported single sign-on and resolved the problems inherent in basic access authentication and digest access authentication supported by existing forward proxy servers. The proxy recognized attributes from shibboleth IdP. The proxy also sets session cookies to control HTTP sessions between a browser and the proxy. The proxy forwards all of requests from the browser to an origin server except for the cookies targeted to the proxy. The system worked well.

As future work, we have to support to proxy of HTTPS and to log individual HTTPS access. The proposed proxy can not log HTTPS because information of target URIs are encrypted between a web browser and a web server. A reverse proxy can it. We will design the proxy by combination of forward proxy for HTTP and reverse proxy for HTTPS.

References

[1] T. Berners-Lee, R. Fielding, and H. Frystyk, "Hypertext Transfer Protocol – HTTP/1.0," RFC 1945 (Informational), Internet Engineering Task Force, May 1996. [Online]. Available: http://www.ietf.org/rfc/rfc1945.txt

[2] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication," RFC 2617 (Draft Standard), Internet Engineering Task Force, Jun. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2617.txt

[3] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1," RFC 2616 (Draft Standard), Internet Engineering Task Force, Jun. 1999, updated by RFCs 2817, 5785. [Online]. Available: http://www.ietf.org/rfc/rfc2616.txt

[4] "Shibboleth documentation," https://wiki.shibboleth.net/.

[5] P. Gauthier, J. Cohen, M. Dunsmuir, , and C. Perkins, "Internet draft: Web proxy auto-discovery protocol (work in progress)," Internet Engineering Task Force, November 2000.

[6] I. Spare, "Deploying the squid proxy server on linux," *Linux J.*, vol. 2001, March 2001. [Online]. Available: http://portal.acm.org/citation.cfm?id=364764.364769

[7] A. Barth, "HTTP State Management Mechanism," RFC 6265 (Proposed Standard), Internet Engineering Task Force, Apr. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6265.txt

[8] "Opensearch," http://www.opensearch.org.